ТРЕЋА БЕОГРАДСКА ГИМНАЗИЈА

Београд, Његошева 15

Матурски рад из Рачунарства и информатике

Најбоље решење за игру „Мој Број”

Ментор:

Иван Симић

Ученик:

Јован Пралица
Одељење: IV/1

Београд, 2022

САДРЖАЈ

1. УВОД

Игра “Мој Број” је једна од игара у ТВ емисији „Слагалица“ која се емитује на програму РТС. Циљ игре је да од 6 задатих бројева од којих прва 4 броја морају да буду једноцифрени бројеви (1-9), а задња 2 броја су бројеви између 10 и 100, с тим што први број мора да буде 10 или 25 а други 50 или 100 и тиме је осигурано да се не може десити да први број буде већи од другог. Такође се додељује и насумично решење тј. насумично изабран број од 50 до 999. Циљ игре је да са основним математичким операцијама (сабирање, одузимање, множење, дељење) и заградама, дође до задатог броја, или ако је то немогуће, до најближег задатог броја. Играчи имају унапред одређено време за решавање израза које је у трајању од 1 минута. Након истека времена играчи, у зависности од тога који играч је на реду, морају да прикажу своју поставку и да прикажу који број су успели да добију. У овом програму не постоји могућност да играју два играча већ само један, с тим што постоје две опције у почетном менију где играч може да изабере да ли жели да самостално унесе бројеве или жели да игра против компјутера и да добије насумичне бројеве. Више о почетном менију можете прочитати на страни 4.

Програм је урађен у програмском језику *Python* и у текст едитору под називом *PyCharm*. Ради на оптимизованој верзији методе под називом „*brute-force*”, с тим што је та метода превише спора па су накнадно додате разне оптимизације како би се побољшало време и тачност решења. Циљ је био да се, ако је икако могуће, тачност решења и брзина доведу до максималних могућих перформанси и комплексност решења се у том случају занемарује. Више о томе можете сазнати на страни 5 на којој је детаљније објашњено шта је то „*brute-force*” и како је имплементиран у овом програму.

У наставку овог рада моћи ћете такође да видите и исечке кода са детаљним објашњењима који део кода има коју примену у програму. Такође су у коду додати и коментари који додатно објашњавају одређене блокове кода чему служе и како раде. Програм у оригиналној форми има близу 500 линија кода али у овом раду ће бити издвојени само неки делови који су најбитнији ради бољег објашњавања и илустрације програма. Комплетан код можете видети на интернет адреси: github.com/JovanPralica.

На самом крају рада ће бити издвојени неки резултати тестирања са различитим резултатима, брзином израчунавања и тачности решења са додатним објашњењима како је програм дошао до одређених решења и зашто. Након резултата ће бити финални закључак о томе зашто сам се баш одлучио за израду програма у *Python*-у, зашто баш „*brute-force*” и одговори на нека од најчешћих питања.

2. КОРИСНИЧКИ МЕНИ

Кориснички мени се састоји из 2 дела. На приказаној слици можемо видети да се он налази скроз на крају кода али место на коме се налази у коду нема толике везе када ће бити приказан, а у овом случају ће кориснички мени бити прва ствар која ће бити приказана играчу.

У менију постоје две опције, прва опција је опција под називом „Решење За Мој Број". То је опција која нам омогућава да ми сами унесемо 6 жељених бројева и решење. Након што смо унели жељене бројеве, уносимо решење. Након тога ће програм у најкраћем могућем року доћи до задатог решења. Битно је да се напомене да је програм направљен на принципу да што брже и што тачније дође до решења. Због тога се може десити да програм на тежи начин дође до неких, на изглед, једноставних решења али је и то донекле оптимизовано убацивањем правила „предности једноставних математичких шаблона" о којима ћу причати мало касније.

```
def menu():
    user_input = int(input('\n1) Resenje Za Moj Broj \n2) Igra Moj Broj \n\n* Unesite broj:'))

    if user_input == 1:
        cheat()
    elif user_input == 2:
        game()
    else:
        print('\n* Mozete uneti samo ponudjene brojeve')
        menu()

menu()
```

Друга опција је „Игра Мој Број" која представља исту игру „Мој Број" као у ТВ емисији „Слагалица" с тим што је ова игра само за једног играча. Важно је рећи да ова игра користи скоро 85% истих блокова кода као и прва опција „Решење За Мој Број" из разлога што обе игре раде на „*brute-force*" методи.

На приложеној слици можемо да видимо да постоји посебна функција под називом „*menu*" у којој се налази код за извршавање првог корисничког уноса са којим се корисник среће. Можемо видети да постоје још две функције под називамо „*cheat*" и „*game*" и оне су означене редним бројевима 1 и 2 тако да играч може да одабере само једну од две понуђене опције.

3. МЕТОДА ПО КОЈОЈ ПРОГРАМ РАДИ

Програм ради на методи под називом „*brute-force*”. Дефиниција „*brute-force*” методе је: „У рачунарству, „*brute-force*” претрага, такође позната као генерисање и тестирање, је веома општа техника решавања проблема и алгоритамска парадигма која се састоји од систематског набрајања свих могућих кандидата за решење и провере да ли сваки кандидат задовољава исказ проблема”. У секцији 3.1 ћу детаљније објаснити како „*brute-force*” метода ради, и како је имплементирана у овом програму.

Дакле, користећи „*brute-force*” методу, ми од већ задатих бројева покушавамо да дођемо до задатог решења испробавајући сваку комбинацију бројева, математичких операција и заграда. Ако бисмо пустили ову методу да ради без додатних оптимизација могли би да чекамо и до неколико дана за комплексније изразе док програм не би дошао до решења. Зато сам увео нешто сто сам назвао „математички шаблони” или „*mathematical patterns*”. Детаљније о математичким шаблонима можете видети у секцији 4.2. То су већ направљени шаблони који већ имају интегрисана места где стоје бројеви, математичке операције и заграде, то изгледа на пример овако:

X = Број
M = Математичка операција
() = Заграде
Шаблон = X M (X M X) на пример 2 * (1 + 2)

Ово значи да увек, кад год се овај шаблон користи, мора да иде прво број, па математичка операција, па број, тиме су елиминисани било какви непотребни покушаји и програм ради за десетине хиљада пута брже, са тим сто се може јавити проблем као на пример, дељење са нулом. Тај проблем је решен преко „*try*” и „*except*” команде у *Python*-у преко којих можемо наредити програму да “игнорише” сваки израз који у себи садржи дељење са нулом. „*Try*” и „*except*” команде су нам помогле да такође решимо проблем ако би играч унео неко слово, или неки број који није из листе понуђених бројева. Овде можемо видети исечак из кода где се „except” користи како би игнорисао свако дељење са нулом и тиме убрзао цео процес.

```
except ZeroDivisionError:
    continue
```

3.1 ДЕТАЉНО ОБЈАШЊЕЊЕ ШТА ЈЕ ТО “BRUTE FORCE”

„*Brute-force*” је један од најпопуларнијих и најпознатијих алгоритамских метода за решавање одређених проблема у програмирању, ради на томе да се проба свака могућа комбинација док се не дође до тачног решења. *„Brute-force“* сам по себи би био веома спор и неефикасан начин решавања проблема ако би се радило о комплекснијим проблемима, зато је обично уз *„Brute-force“* методу намештена и оптимизација. Оптимизације за „*brute-force*” методу значе да ће се на било који начин „скратити” пут и време до решавања проблема имплементирањем одређених правила или смерница по којима ће „*brute-force*” функционисати, које ће аутоматски елиминисати неке покушаје који немају смисла и тиме добити на времену, док би тачност остала иста.

Неке оптимизације које сам додао у овом програму су:

- Математички шаблони
- Метод паметног распореда шаблона
- Тајмер
- Правила за игнорисање математички немогућих израза

Свака од наведених оптимизација је детаљно објашњења у наставку рада. Објаснићу укратко неке од ових метода како би нам било јасније како се могу имплементирати оптимизације у „*brute-force*”. Битно је да се напомене да је свака од ових метода имплементирана у једну од две сврхе:

- Побољшања брзине
- Побољшања тачности

Побољшање брзине се постизало тако што би помоћу математичких шаблона и одређеног распореда шаблона могли да убрзамо цео процес „*brute-force*”, тако што би програм тачно знао којим „путем” да решава задати израз.

Тајмер је имплементиран како би представљао неку врсту „усмерења” за наставак рада програма када би прошло одређено време од извршења одређених функција. Како би се максимално побољшала брзина а да се не оштети тачност, направљени су и шаблони за тајмер по којима тајмер након истека одређеног времена извршава одређене радње.

3.2 ИЗБАЦИВАЊЕ БРОЈЕВА И СОРТИРАЊЕ

Ова секција је везана искључиво за други део програма „Игра Мој Број”. Први корак за започињање игре је било да се из две листе, насумично изаберу бројеви у правилима описаним у секцији 1. Ти бројеви би се затим убацили у „*nested list*”, који ће у себи имати у једној половини бројеве које је компјутер избацио, а у другој половини бројеве које је играч уписао (слика 1). О томе зашто постоје две листе ћу објаснити управо сада објашњавањем филтера који је такође имплементиран у овом делу програма.

Други део сортирања је био и најзахтевнији, потребно је да се цео „*input*” који играч унесе убаци у једну листу, а затим сортира. Први део сортирања је да се елиминишу сви елементи који нису бројеви, ово је битно због тога што ће након филтрирања само бројева, две листе ће моћи да се упореде да ли су исте. Ако су исте, значи да је играч унео само постојаће бројеве, а ако нису исте то онда значи да је играч унео бројеве који нису доступни. Након сортирања две листе, ако би се утврдило да је играч унео нешто погрешно, програм би престао са радом и искочила би грешка „Можете унети само постојаће бројеве”. Ако се деси да је играч унео само бројеве који су били понуђени, програм ће одмах избацити корисников израз и решење тог израза док ће у позадини радити на решавању истог израза на свој начин.

Слика 1. Избацивање насумичних бројева

```
# FIRST FOUR SELECTION
for num in firstFour:
    num1 = random.choice(firstFour)
    num2 = random.choice(firstFour)
    num3 = random.choice(firstFour)
    num4 = random.choice(firstFour)

# LAST TWO SELECTION
for lastNum in lastTwo:
    num5 = random.choice(firstTwo)
    num6 = random.choice(lastTwo)

# MB LISTS
mbList = [[num1,num2,num3,num4,num5,num6], []]
mbSol = int(random.randint(50, 999))
mbOps = ['+', '-', '*', '/']
```

4. ИНТЕГРИСАЊЕ ПРАВИЛА ЗА КОРИШЋЕЊЕ

Интегрисање правила за коришћење је један од кључних фактора да би програм уопште радио онако тачно како ради у игри „Мој Број" и да на одређени начин долази до решења. У секцији 4 ћу детаљније објаснити следеће појмове:

- Метод "Левка"
- Математички шаблони
- Паметан распоред математичких шаблона
- Тајмер

Свака од ових метода је примењена у сврху убрзања И побољшања тачности самог програма. Овде ћу укратко објаснити шта свака од ових метода ради и како су оне међусобно повезане.

Метод "Левка" је метод елиминације бројева који се користи током распоређивања бројева у другом делу програма. Метод ради тако што, када се један број насумично одабере да буде коришћен у игри, он га елиминише из листе како не би могао опет да се понови. Ова радња се обавља након сваког насумичног узимања бројева, где се број бројева у оптицају који су могући да се извуку смањује до тренутка док не остане само један број, и тиме се овај метод завршава.

Математички шаблони су шаблони који су већ унапред интегрисани и који уз помоћ пермутације бројева и основних рачунских операција брже и лакше долазе до решења. Унапред је испрограмирано да када се у шаблонима појави 0 или дељење са 0, одмах се прескаче и пребацује на следећи шаблон. Исти случај је са свим немогућим изразима. Ово је био само пример како шаблони раде. Када шаблони не би постојали, програму би много дуже требало да дође до решења, него када има тачан пут по којем иде.

Паметни распоред математичких шаблона је коришћење чињенице да програм чита и извршава код одозго на доле, и према томе су шаблони распоређени тако да програм брже дође до решења.

Тајмер је имплементиран као још једна додатна функција другог дела програма. Она служи да играч, ако жели, може да ограничи време за извршавање израза. Такође постоји тајмер који мери време рада програма и према времену које је истекло мења „временске шаблоне" како би брже дошао до тачнијег решења, као и да не би могло да се деси да програм ради без престанка.

4.1 МЕТОД ЛЕВКА

Метод левка је један од најбољих, и најбржих начина да се у оба дела програма бројеви распореде без понављања и са максималном тачношћу. Метод левка у првом делу програма ради тако што када играч унесе бројеве и решење које жели да добије са унетим бројевима, метод левка ће распоређивати унете бројеве на 6 варијабли, где неће постајати могућност да се број понови, јер чим се број распореди у неку од варијабли, он се аутоматски брише из листе. Метод се извршава 5 пута све док не остане само један број. Када остане само један број он се распоређује у варијаблу која је остала.

```
# FUNNEL METHOD

num1 = random.choice(array)
array.remove(num1)
num2 = random.choice(array)
array.remove(num2)
num3 = random.choice(array)
array.remove(num3)
num4 = random.choice(array)
array.remove(num4)
num5 = random.choice(array)
array.remove(num5)
num6 = array[0]
array.remove(num6)

# RE-FILLING ARRAY
array.append(num1)
array.append(num2)
array.append(num3)
array.append(num4)
array.append(num5)
array.append(num6)
```

Након што се сви бројеви распореде, бројеви ће се вратити у листу како би могли опет да се користе ако има потребе. Позивање листе ће у 99,99% случајева морати да се обави више пута, јер када год програм преко математичких шаблона буде покушао да дође до решења и не буде успео, он ће опет урадити цео процес распоређивања бројева. Ово је битно, јер када се бројеви распореде више пута, биће више пута распоређени по различитим варијаблама, што омогућава пермутацију бројева и брже долажење до задатог решења. На слици изнад можете да видите како метод левка изгледа у коду.

4.2 МАТЕМАТИЧКИ ШАБЛОНИ

Математички шаблони су шаблони који садрже одређене, унапред направљене, математичке комбинације основних математичких операција, понуђених бројева, и заграда. Циљ математичких шаблона је да убрзају и олакшају алгоритму у тражењу решења од понуђених бројева. Проблем који се може јавити јесте да може да се деси да за одређену комбинацију бројева и решења које треба да се добије не постоји математички шаблон који може довести до тог решења, зато је убачено онолико шаблона колико је оптимално потребно како би програм имао најбољу пропорцију брзине и тачности јер се ни у самој игри не може увек добити тачно решење од понуђених бројева.

Шаблони су направљени од оних најлакших који садрже само 2 броја, до оних најкомплекснијих који садрже свих 6 бројева и комплексне шаблоне заграда. Најкомплекснијих шаблона има и највише због тога што број бројева који се налази у шаблону доприноси томе да је могуће направити много више варијација са заградама, математичким операцијама, пермутацијама чланова итд.

Шаблони су у првобитној верзији били поређани од најлакших ка најтежима, али имплементацијом паметног распореда шаблона се може доста добити на времену док би тачност остала иста.

Овде можемо видети листу свих шаблона који су имплементирани у коду. У следећој секцији ћемо детаљније видети како тачно шаблони изгледају и њихов распоред.

```
# PATTERN LIST

pattern_list = [easy_pattern1,easy_pattern2,easy_pattern3,pattern1, pattern2, pattern3, pattern4,
                pattern5, pattern6, pattern7, pattern8, pattern9,pattern10, pattern11, pattern12,
                pattern13, pattern14, pattern15, pattern16, pattern17, pattern18, pattern19, pattern20,
                pattern21, pattern22, pattern23, pattern24,pattern25, pattern26, pattern27,
                pattern28, pattern29, pattern30]
```

4.3 МЕТОД ПАМЕТНОГ РАСПОРЕДА ШАБЛОНА

Метод паметног распореда шаблона је метод који ради на основном правилу програмирања, а то је да који год да је програм у питању, код се увек чита одозго на доле. Као што је већ речено у претходној секцији, првобитна верзија програма је имала распоред шаблона од најлакшег до најтежег и то је радило, али се имплементацијом паметног распореда шаблона доста увећала брзина.

Прва три шаблона су најлакши шаблони, који чине максимално 3 или 4 броја без компликованих заграда, они су у коду названи „*easy patterns*" и убачени су да ако се деси да је израз јако једноставан, програм одмах проба да реши израз са лакшим шаблонима како би добио брже и лепше решење.

Ако програм не успе да нађе решење са лакшим шаблонима, онда прелази на оне нормалне који су распоређени тако да иде прво лакши онда тежи, и тако су распоређени све до краја. Циљ тога је да шанса за сваки пар шаблона буде већа јер би у једном пару ишао лакши и тежи шаблон и тиме би се добило на времену јер, невезано да ли је израз лакши или тежи, постојала би већа шанса да се пре дође до решења ако би шаблони били постављени по оваквом редоследу. У први пар су постављени најлакши шаблон (који је ипак компликованији од сета „*easy patterns*" шаблона) и најтежи шаблон у целом програму. Затим је у другом пару други најлакши и други најтежи и низ се тако наставља све док се не направе сви парови са лакшим и тежим шаблонима.

```
# SMART PATTERN PLACEMENT

easy_pattern1 = f'{num1} {random.choice(mbOps)} {num2}'
easy_pattern2 = f'{num1} {random.choice(mbOps)} {num2} {random.choice(mbOps)} {num3}'
easy_pattern3 = f'{num1} {random.choice(mbOps)} {num2} {random.choice(mbOps)} {num3} {random.choice(mbOps)} {num4}

pattern1 = f'{num1} {random.choice(mbOps)} ({num2} {random.choice(mbOps)} {num3} {random.choice(mbOps)} {num4})'
pattern30 = f'({num1} {random.choice(mbOps)} {num2} {random.choice(mbOps)} ({num3} {random.choice(mbOps)} {num4})
```

5. ИЗБАЦИВАЊЕ РЕЗУЛТАТА И ТАЈМЕР

Задњи део програма је само избацивање резултата тј. избацивање поступка до задатог решења од задатих бројева. Избацивање решења је другачије у ова два дела програма. У првом делу програма, где играч задаје бројеве и решење које треба да се добије, решење се одмах избацује у конзолу чим програм заврши са радом и ту рад целог програма престаје. Други део програма, где играч игра против машине која му задаје бројеве, решење се избацује тек након што играч унесе своје решење, или ако се играч одлучио да игра са ограниченим временом, након што време истекне.

Тајмер или временско ограничење, је убачено у програм и има више различитих примена, прва примена је да се ограничи време ако играч одлучи да игра са ограниченим временом против компјутера, друга примена је у самој *„brute-force“* методи, где су имплементирани „временски шаблони”. Временски шаблони су шаблони који ограничавају колико ће се временски програм задржати на одређеном математичком шаблону тј. колико ће времена потрошити на одређени сет шаблона док не пређе на следећи и тиме повећа брзину долажења до решења или до најближег решења. Некад се може десити да се не може доћи задатим бројевима до задатог решења, у том случају, након истека задњег временског шаблона, програм ће избацити поруку у конзолу „Програм није успео да нађе приближно решење”. Самим тим се елиминише могућност да се деси да програм тражи одређено решење до бесконачности.

```
elif timer >= 5 and pattern != mbSol:

    if eval(pattern) == mbSol:
        print('\n' + '* Rezultat:', pattern, '=', mbSol)
        print('\n' + f'* Vreme programa: {timer} sec')
        time.sleep(20)
        sys.exit()
```

Оба тајмера су комбинована са „threading” модулом који је један од базичних модула за *Python*. *„Threading“* нам омогућава да можемо да покрећемо одређене блокове кода да раде у позадини док се неки други блокови кода извршавају у исто време, с тим што ми *„threading“* блокове кода не можемо да видимо јер се они извршавају у позадини. Самим тим ми не можемо физички видети промену временских шаблона јер, док програм преко *„brute-force“* методе тражи решење преко математичких шаблона, временски шаблони се у позадини програма смењују и преусмеравају ток рада програма са једног сета шаблона на други.

6. РЕЗУЛТАТИ ТЕСТИРАЊА

У овој секцији ће бити приказани неки примери тестирања програма. Обзиром да се исти алгоритам користи и у једном и у другом делу игре, користићемо само први део програма ради лакшег тестирања.

```
* Unesite broj:1

* Unesite brojeve: 1,2,3,4,5,6

* Unesite resenje: 123

* Rezultat: 3 + 4 * 6 * 5 = 123

* Vreme programa: 0.125 sec
```

У првом тесту можемо видети 6 насумично убачених бројева и насумично решење које је програм успео да нађе за 0.12 секунди.

```
* Unesite broj:1

* Unesite brojeve: 2,1,2,1,10,25

* Unesite resenje: 590

* Rezultat: (2 * 10 + 1) * ((2 + 25) + 1) = 588

* Vreme programa: 5.53125 sec
```

На другој слици можемо видети да је програму требало знатно дуже времена да дође до приближног решења јер са постојећим бројевима није успео да дође до тачног задатог броја. Овде се може видети пример временског шаблона где је програм после 5 секунди одустао од тражења тачног решења и прешао на приближна решења.

```
* Unesite broj:1

* Unesite brojeve: 1,1,1,1,1,1

* Unesite resenje: 590

* Program nije uspeo da nadje priblizno resenje :(
```

На трећој слици видимо трећи случај где програм није нашао приближно решење.

7. ЗАКЉУЧАК

Сам програм што се тиче програмирања није био много захтеван. Најзахтевнији део целог овог пројекта је била сама идеја на који начин доћи до решења. Једина два начина на која ја мислим да се може доћи до решења су прављење вештачке интелигенције која би симулирала *„brute-force“* само на много паметнији начин, а не чистим алгоритмом већ боље осмишљеним шаблонима који би се мењали током времена. Други начин, који је и искоришћен у овом програму, је прављење *„brute-force“* алгоритма који би по одређеним шаблонима уз помоћ пермутације чланова и основних математичких операција и заграда испробавао више шаблона у исто време и зауставио са радом чим би се појавио тачан поступак са којим се од задатих бројева може доћи до решења.

Сам *„brute-force“* није био довољан. Многе оптимизације су морале да буду убачене како би се избегли неки од главних проблема као што су:

- Дељење са нулом
- Неизједначене заграде
- Превише времена за израчунавање израза
- Правила саме игре (на пример, да играч не може да унесе било који број)

Неки од необичних проблема су били препуњавање *RAM* меморије и активација анти-вируса због одређених модула који су били убачени.

Threading модул је модул у *Python*-у који нам омогућава да покренемо више функција у позадини, што је резултирало да се *RAM* меморија понекад препуни и некада се дешавало да програм не дође до тачног решења, а када би се направила пауза између тестирања од пар минута, дошао би до истог тог решења за мање од 1 секунде.

8. ЛИТЕРАТУРА

1. Brute-force method: https://en.wikipedia.org/wiki/Brute-force_search

Датум предаје: _____

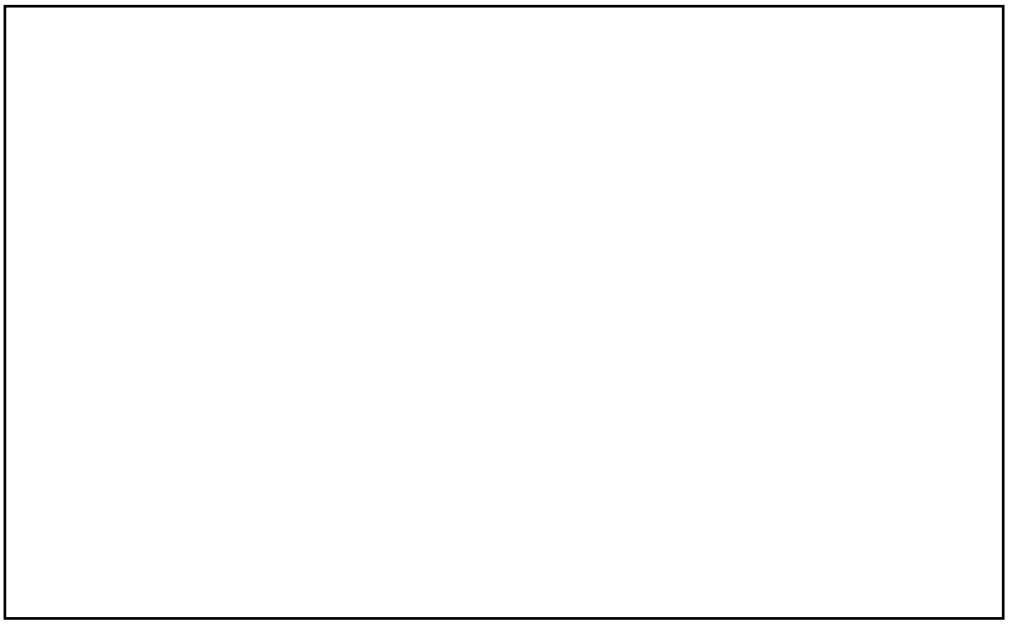
Комисија:

Председник _____

Испитивач _____

Члан _____

Коментар:



Датум одбране: _____ Оцена: _____(_____)